| | |
|---|---|
| **Experiment Number** | 6 |
| **Experiment Title** | Experiment 6: Web Scraper using LLMs |
| **Date of Experiment** | 13/03/2025 |
| **Date of Submission** | 20/03/2025 |

**Experiment 6: Web Scraper using LLMs**

**Objective:**
**To create a web scraper application integrated with LLMs for processing scraped data.**
**Detailed Procedure:**
**1. Use Python libraries like BeautifulSoup and Requests to scrape web data.**
**2. You can also use LlamaIndex for Web Scraping and Ollama for open ended LLMs**
**2. Integrate LLMs to process and summarize the scraped information.**
**3. Develop a Flask backend for handling scraping tasks and queries.**
**4. Create an HTML/CSS frontend to initiate scraping (like the web page to scrape) and**
**display results.**
**5. You can also take a topic and search the web for a web page and then scrape it.**

**Code:- From next page as follows**
**app.py, llama_processor.py, scraper.py, search.py, index.html, style.css, script.js**

```python
from flask import Flask, render_template, request, jsonify
from modules.scraper import WebScraper
from modules.llm_processor import LLMProcessor
from modules.search import WebSearch

app = Flask(__name__)

# Initialize modules
scraper = WebScraper()
llm = LLMProcessor()
search = WebSearch()

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/scrape', methods=['POST'])
def scrape_url():
    data = request.json
    url = data.get('url')
    scrape_method = data.get('method', 'bs4')  # Default to BeautifulSoup

    if not url:
        return jsonify({"error": "URL is required"}), 400

    try:
        # Scrape content based on the selected method
        if scrape_method == 'llamaindex':
            result = scraper.scrape_with_llamaindex(url)
        else:
            result = scraper.scrape_with_bs4(url)

        # Check if scraping was successful
        if not result.get('content'):
            return jsonify({"error": result.get('error', 'Failed to scrape content')}), 400

        return jsonify(result)

    except Exception as e:
        return jsonify({"error": str(e)}), 500

@app.route('/process', methods=['POST'])
def process_content():
    data = request.json
    content = data.get('content')
    processing_type = data.get('type', 'summarize')  # Default to summarize

    if not content:
        return jsonify({"error": "Content is required"}), 400

    try:
        # Process content based on the selected processing type
        if processing_type == 'keypoints':
            result = llm.extract_key_points(content)
        elif processing_type == 'sentiment':
            result = llm.analyze_sentiment(content)
        else:
            result = llm.summarize(content)

        return jsonify({"result": result})

    except Exception as e:
        return jsonify({"error": str(e)}), 500

@app.route('/search', methods=['POST'])
def search_topic():
    data = request.json
    topic = data.get('topic')

    if not topic:
        return jsonify({"error": "Topic is required"}), 400

    try:
        # Get URLs for the topic
        urls = search.get_urls_for_topic(topic)

        if not urls:
            return jsonify({"error": "No URLs found for the topic"}), 404

        return jsonify({"urls": urls})

    except Exception as e:
        return jsonify({"error": str(e)}), 500

@app.route('/scrape-and-process', methods=['POST'])
def scrape_and_process():
    data = request.json
    url = data.get('url')
    topic = data.get('topic')
    processing_type = data.get('processingType', 'summarize')

    # Either URL or topic must be provided
    if not url and not topic:
        return jsonify({"error": "Either URL or topic is required"}), 400

    try:
        # If topic is provided but URL is not, search for URLs
        if topic and not url:
            urls = search.get_urls_for_topic(topic)
            if not urls:
                return jsonify({"error": f"No URLs found for topic: {topic}"}), 404
            url = urls[0]  # Use the first URL

        # Scrape the content
        result = scraper.scrape_with_bs4(url)

        if not result.get('content'):
            return jsonify({"error": result.get('error', 'Failed to scrape content')}), 400

        # Process the content based on the selected type
        if processing_type == 'keypoints':
            processed_result = llm.extract_key_points(result['content'])
        elif processing_type == 'sentiment':
            processed_result = llm.analyze_sentiment(result['content'])
        else:
            processed_result = llm.summarize(result['content'])

        return jsonify({
            "url": url,
            "title": result.get('title', 'No title'),
            "domain": result.get('domain', ''),
            "scraped_content": result['content'][:500] + "...",  # Preview of content
            "processed_result": processed_result
        })

    except Exception as e:
        return jsonify({"error": str(e)}), 500

if __name__ == '__main__':
    app.run(debug=True)

```

```python
import requests

class LLMProcessor:
    def __init__(self, model="llama2"):
        """Initialize the LLM processor with the specified model"""
        self.model = model
        self.ollama_url = "http://localhost:11434/api"
        self._ensure_model_available()

    def _ensure_model_available(self):
        """Check if the model is available and pull it if needed"""
        try:
            # Check if Ollama is running
            response = requests.get(f"{self.ollama_url}/tags")

            # Check if our model is in the list
            models = response.json().get("models", [])
            model_exists = any(m["name"] == self.model for m in models)

            if not model_exists:
                print(f"Model {self.model} not found. Pulling from Ollama...")
                self._pull_model()

        except requests.RequestException:
            print("Ollama service not detected. Please ensure Ollama is installed and running.")
            print("Visit https://ollama.ai/ for installation instructions.")

    def _pull_model(self):
        """Pull the model from Ollama"""
        try:
            response = requests.post(
                f"{self.ollama_url}/pull",
                json={"name": self.model}
            )

            if response.status_code == 200:
                print(f"Successfully pulled {self.model}")
            else:
                print(f"Failed to pull {self.model}: {response.text}")
        except Exception as e:
            print(f"Error pulling model: {str(e)}")

    def process_content(self, content, instruction):
        """Process content with the LLM based on the given instruction"""
        try:
            prompt = f"""
            {instruction}

            Content to process:
            {content[:10000]}  # Limiting to first 10K chars to avoid token limits
            """

            response = requests.post(
                f"{self.ollama_url}/generate",
                json={
                    "model": self.model,
                    "prompt": prompt,
                    "stream": False
                }
            )

            if response.status_code == 200:
                result = response.json()
                return result.get("response", "No response generated")
            else:
                return f"Error: {response.status_code} - {response.text}"

        except Exception as e:
            return f"Error processing content: {str(e)}"

    def summarize(self, content):
        """Summarize the provided content"""
        instruction = "Please provide a concise summary of the following content:"
        return self.process_content(content, instruction)

    def extract_key_points(self, content):
        """Extract key points from the provided content"""
        instruction = "Please extract the main points and key information from the following content:"
        return self.process_content(content, instruction)

    def analyze_sentiment(self, content):
        """Analyze the sentiment of the provided content"""
        instruction = "Please analyze the sentiment of the following content. Is it positive, negative, or neutral? Explain your reasoning:"
        return self.process_content(content, instruction)

```

```python
import requests
from bs4 import BeautifulSoup
from llama_index.readers.web import TrafilaturaWebReader
import re
from urllib.parse import urlparse

class WebScraper:
    def __init__(self):
        self.session = requests.Session()
        self.session.headers.update({
            'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36'
        })

    def scrape_with_bs4(self, url):
        """Scrape web content using BeautifulSoup"""
        try:
            response = self.session.get(url, timeout=10)
            response.raise_for_status()

            soup = BeautifulSoup(response.content, 'html.parser')

            # Remove script and style elements
            for script in soup(["script", "style"]):
                script.extract()

            # Get text content
            text = soup.get_text(separator='\n')

            # Clean up text: remove extra newlines and whitespace
            clean_text = re.sub(r'\n+', '\n', text).strip()
            clean_text = re.sub(r'\s+', ' ', clean_text)

            # Extract title
            title = soup.title.string if soup.title else "No title found"

            # Extract domain
            domain = urlparse(url).netloc

            return {
                "url": url,
                "title": title,
                "domain": domain,
                "content": clean_text
            }
        except Exception as e:
            return {
                "url": url,
                "error": str(e),
                "content": None
            }

    def scrape_with_llamaindex(self, url):
        """Scrape web content using LlamaIndex's SimpleWebPageReader"""
        try:
            SimpleWebPageReader = download_loader("SimpleWebPageReader")
            loader = SimpleWebPageReader()
            documents = loader.load_data(urls=[url])

            if documents and len(documents) > 0:
                return {
                    "url": url,
                    "title": documents[0].metadata.get("title", "No title found"),
                    "domain": urlparse(url).netloc,
                    "content": documents[0].text
                }
            else:
                return {
                    "url": url,
                    "error": "No content found",
                    "content": None
                }
        except Exception as e:
            return {
                "url": url,
                "error": str(e),
                "content": None
            }

```

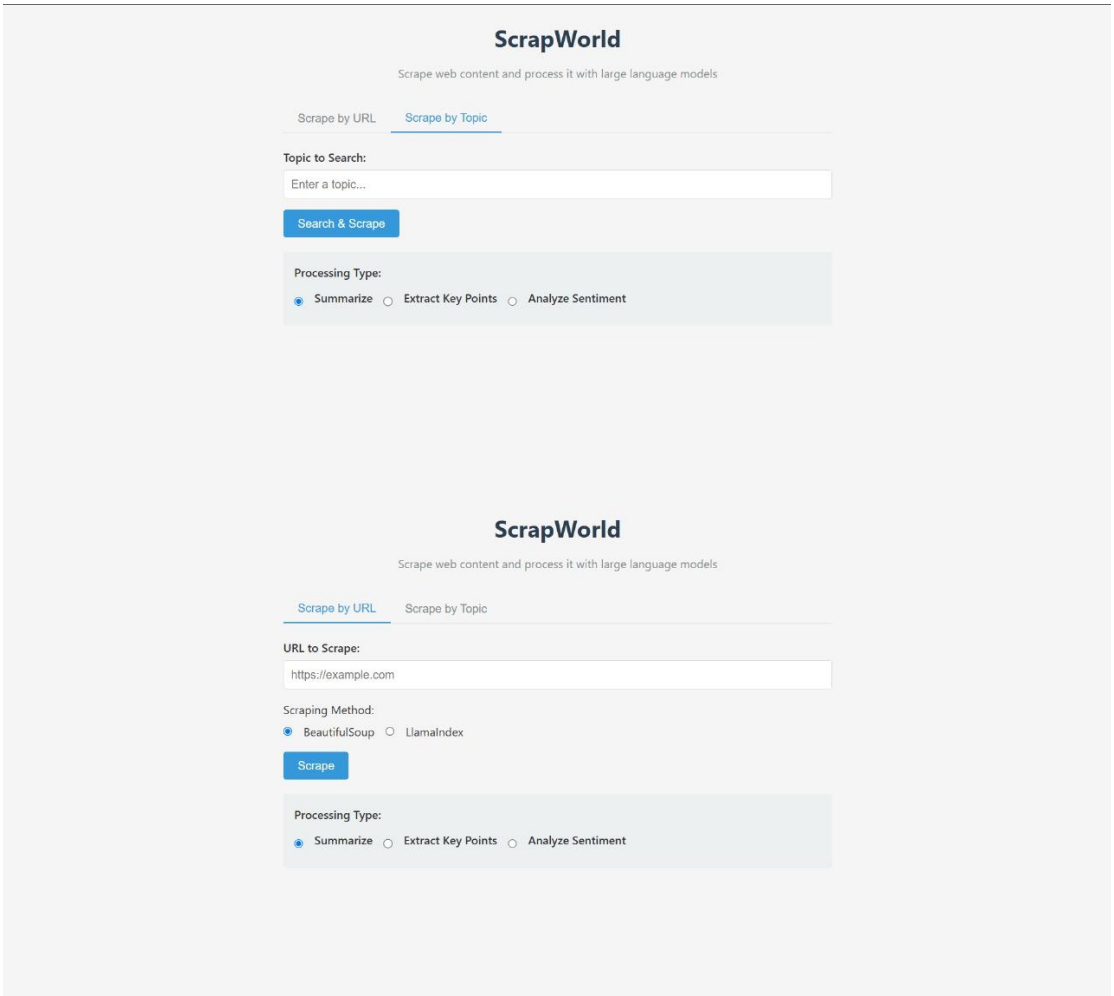
```python
import requests
from bs4 import BeautifulSoup
from urllib.parse import quote_plus

class WebSearch:
    def __init__(self):
        self.session = requests.Session()
        self.session.headers.update({
            'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36'
        })

    def search_google(self, query, num_results=5):
        """
        Search Google for the query and return a list of result URLs.
        This is a basic implementation and might be blocked by Google.
        For production, consider using a proper search API.
        """
        try:
            search_url = f"https://www.google.com/search?q={quote_plus(query)}&num={num_results}"
            response = self.session.get(search_url, timeout=10)
            response.raise_for_status()

            soup = BeautifulSoup(response.text, 'html.parser')

            # Find all result links
            search_results = []
            for result in soup.select('div.g div.yuRUbf a'):
                url = result['href']
                if url.startswith('http') and 'google.com' not in url:
                    search_results.append(url)

            return search_results[:num_results]

        except Exception as e:
            print(f"Error searching: {str(e)}")
            return []

    def get_urls_for_topic(self, topic, num_results=5):
        """Get a list of relevant URLs for a given topic"""
        return self.search_google(topic, num_results)

```

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Web Scraper with LLM</title>
    <link rel="icon" type="image/x-icon" href="{{ url_for('static', filename='favicon.ico') }}">
    <link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}">
</head>
<body>
    <div class="container">
        <header>
            <h1>ScrapWorld</h1>
            <p>Scrape web content and process it with large language models</p>
        </header>

        <div class="tabs">
            <button class="tab-btn active" data-tab="url-tab">Scrape by URL</button>
            <button class="tab-btn" data-tab="topic-tab">Scrape by Topic</button>
        </div>

        <div class="tab-content active" id="url-tab">
            <div class="input-group">
                <label for="url-input">URL to Scrape:</label>
                <input type="text" id="url-input" placeholder="https://example.com">
            </div>

            <div class="options-group">
                <label>Scraping Method:</label>
                <div class="radio-group">
                    <input type="radio" id="bs4-method" name="scrape-method" value="bs4" checked>
                    <label for="bs4-method">BeautifulSoup</label>

                    <input type="radio" id="llamaindex-method" name="scrape-method" value="llamaindex">
                    <label for="llamaindex-method">LlamaIndex</label>
                </div>
            </div>

            <button id="scrape-url-btn" class="primary-btn">Scrape</button>
        </div>

        <div class="tab-content" id="topic-tab">
            <div class="input-group">
                <label for="topic-input">Topic to Search:</label>
                <input type="text" id="topic-input" placeholder="Enter a topic...">
            </div>
            <button id="search-topic-btn" class="primary-btn">Search & Scrape</button>
        </div>

        <div class="processing-options">
            <label>Processing Type:</label>
            <div class="radio-group">
                <input type="radio" id="summarize" name="processing-type" value="summarize" checked>
                <label for="summarize">Summarize</label>

                <input type="radio" id="keypoints" name="processing-type" value="keypoints">
                <label for="keypoints">Extract Key Points</label>

                <input type="radio" id="sentiment" name="processing-type" value="sentiment">
                <label for="sentiment">Analyze Sentiment</label>
            </div>
        </div>

        <div class="results-container" id="results-container" style="display: none;">
            <div class="loading" id="loading">
                <div class="spinner"></div>
                <p>Processing... This may take a moment</p>
            </div>

            <div class="results" id="results">
                <div class="result-meta">
                    <h2 id="result-title">Title</h2>
                    <p id="result-url">URL: <a href="#" target="_blank"></a></p>
                </div>

                <div class="result-tabs">
                    <button class="result-tab-btn active" data-result-tab="processed">Processed Content</button>
                    <button class="result-tab-btn" data-result-tab="raw">Raw Content</button>
                </div>

                <div class="result-tab-content active" id="processed-content">
                    <pre id="processed-result"></pre>
                </div>

                <div class="result-tab-content" id="raw-content">
                    <pre id="raw-result"></pre>
                </div>
            </div>
        </div>
```

```css
/* Base styles */
* {
    box-sizing: border-box;
    margin: 0;
    padding: 0;
}

body {
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
    line-height: 1.6;
    color: #333;
    background-color: #f5f5f5;
}

.container {
    max-width: 800px;
    margin: 0 auto;
    padding: 20px;
}

/* Header */
header {
    text-align: center;
    margin-bottom: 30px;
}

header h1 {
    color: #2c3e50;
    margin-bottom: 10px;
}

header p {
    color: #7f8c8d;
}

/* Tabs */
.tabs {
    display: flex;
    margin-bottom: 20px;
    border-bottom: 1px solid #ddd;
}

.tab-btn {
    padding: 10px 20px;
    border: none;
    background: none;
    cursor: pointer;
    font-size: 16px;
    color: #7f8c8d;
    transition: all 0.3s ease;
}

.tab-btn.active {
    color: #3498db;
    border-bottom: 2px solid #3498db;
}

.tab-content {
    display: none;
    margin-bottom: 20px;
}

.tab-content.active {
    display: block;
}

/* Form elements */
.input-group {
    margin-bottom: 15px;
}

.input-group label {
    display: block;
    margin-bottom: 5px;
    font-weight: 500;
}

.input-group input {
    width: 100%;
    padding: 10px;
    border: 1px solid #ddd;
    border-radius: 4px;
    font-size: 16px;
}

.options-group {
    margin-bottom: 15px;
}

.radio-group {
    display: flex;
    gap: 15px;
    margin-top: 5px;
}

/* Buttons */
.primary-btn {
    background-color: #3498db;
    color: white;
    border: none;
    padding: 10px 20px;
    font-size: 16px;
    border-radius: 4px;
    cursor: pointer;
    transition: background-color 0.3s ease;
}

.primary-btn:hover {
    background-color: #2980b9;
}

/* Processing options */
.processing-options {
    margin-bottom: 20px;
    padding: 15px;
    background-color: #ecf0f1;
    border-radius: 4px;
}

.processing-options label {
    font-weight: 500;
    margin-bottom: 10px;
    display: block;
}

/* Results container */
.results-container {
    margin-top: 30px;
    border: 1px solid #ddd;
    border-radius: 8px;
    overflow: hidden;
}

.loading {
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content: center;
    padding: 40px;
}

.spinner {
    border: 4px solid rgba(0, 0, 0, 0.1);
    border-left-color: #3498db;
    border-radius: 50%;
    width: 40px;
    height: 40px;
    animation: spin 1s linear infinite;
    margin-bottom: 15px;
}

@keyframes spin {
    0% { transform: rotate(0deg); }
    100% { transform: rotate(360deg); }
}

.results {
    display: none;
```

```
document.addEventListener('DOMContentLoaded', function() {
    // Tab switching
    const tabBtns = document.querySelectorAll('.tab-btn');
    const tabContents = document.querySelectorAll('.tab-content');

    tabBtns.forEach(btn => {
        btn.addEventListener('click', () => {
            // Deactivate all tabs
            tabBtns.forEach(b => b.classList.remove('active'));
            tabContents.forEach(c => c.classList.remove('active'));

            // Activate selected tab
            btn.classList.add('active');
            const tabId = btn.getAttribute('data-tab');
            document.getElementById(tabId).classList.add('active');
        });
    });

    // Result tab switching
    const resultTabBtns = document.querySelectorAll('.result-tab-btn');
    const resultTabContents = document.querySelectorAll('.result-tab-content');

    resultTabBtns.forEach(btn => {
        btn.addEventListener('click', () => {
            // Deactivate all result tabs
            resultTabBtns.forEach(b => b.classList.remove('active'));
            resultTabContents.forEach(c => c.classList.remove('active'));

            // Activate selected result tab
            btn.classList.add('active');
            const tabId = btn.getAttribute('data-result-tab') + '-content';
            document.getElementById(tabId).classList.add('active');
        });
    });

    // Scrape URL button click handler
    document.getElementById('scrape-url-btn').addEventListener('click', () => {
        const url = document.getElementById('url-input').value.trim();
        if (!url) {
            alert('Please enter a URL to scrape');
            return;
        }

        const scrapeMethod = document.querySelector('input[name="scrape-method"]:checked').value;
        const processingType = document.querySelector('input[name="processing-type"]:checked').value;

        // Show loading state
        showLoading();

        // Make API call to scrape and process
        fetch('/scrape-and-process', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify({
                url: url,
                scrapeMethod: scrapeMethod,
                processingType: processingType
            })
        })
        .then(response => response.json())
        .then(data => {
            if (data.error) {
                throw new Error(data.error);
            }
            displayResults(data);
        })
        .catch(error => {
            hideLoading();
            alert('Error: ' + error.message);
        });
    });

    // Search topic button click handler
    document.getElementById('search-topic-btn').addEventListener('click', () => {
        const topic = document.getElementById('topic-input').value.trim();
        if (!topic) {
            alert('Please enter a topic to search');
            return;
        }

        const processingType = document.querySelector('input[name="processing-type"]:checked').value;

        // Show loading state
        showLoading();

        // Make API call to search, scrape and process
        fetch('/scrape-and-process', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify({
                topic: topic,
                processingType: processingType
            })
        })
        .then(response => response.json())
        .then(data => {
            if (data.error) {
                throw new Error(data.error);
            }
            displayResults(data);
        })
        .catch(error => {
            hideLoading();
            alert('Error: ' + error.message);
        });
    });

    // Function to display results
    function displayResults(data) {
        // Hide loading indicator
        hideLoading();

        // Display results container
        document.getElementById('results-container').style.display = 'block';

        // Fill in result data
        document.getElementById('result-title').textContent = data.title || 'No Title';
        const urlElement = document.querySelector('#result-url a');
        urlElement.textContent = data.url;
        urlElement.href = data.url;

```

4. Results/Output:- Entire Screen Shot including Date & Time




5. Remarks:-

Signature of the Student
Apratim Dutta
(Name of the Student)

Signature of the Lab Coordinator

(Name of the Coordinator)